Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática

Search Strategies in Unstructured Overlays

Pedro Marques Fonseca

Mestrado em Engenharia Informática

2008

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática

Search Strategies in Unstructured Overlays

Pedro Marques Fonseca

DISSERTAÇÃO

Projecto orientado pelo Prof. Dr. Hugo Miranda

Mestrado em Engenharia Informática

2008

This work was partially supported by LaSIGE through the FCT
Pluriannual Funding Programme

Declaração

Pedro Fonseca, aluno nº 29837 da Faculdade de Ciências da Universidade de Lisboa, declara ceder os seus direitos de cópia sobre o seu Relatório de Projecto em Engenharia Informática, intitulado "Search Strategies in Unstructured Overlays", realizado no ano lectivo de 2007/2008 à Faculdade de Ciências da Universidade de Lisboa para o efeito de arquivo e consulta nas suas bibliotecas e publicação do mesmo em formato electrónico na Internet.

FCUL, 30 de Junho de 2008

Hugo Miranda, supervisor do projecto de *Pedro Fonseca*, aluno da Faculdade de Ciências da Universidade de Lisboa, declara concordar com a divulgação do Relatório do Projecto em Engenharia Informática, intitulado "Search Strategies in Unstructured Overlays".

Lisboa, 30 de Junho de 2008

Abstract

Unstructured peer-to-peer networks have a low maintenance cost, high resilience and tolerance to the continuous arrival and departure of nodes. In these networks search is usually performed by flooding, which generates a high number of duplicate messages. To improve scalability, unstructured overlays evolved to a two-tiered architecture where regular nodes rely on special nodes, called supernodes or superpeers, to locate resources, thus reducing the scope of flooding based searches. While this approach takes advantage of node heterogeneity, it makes the overlay less resilient to accidental and malicious faults, and less attractive to users concerned with the consumption of their resources and who may not desire to commit additional resources that are required by nodes selected as superpeers.

Another point of concern is churn, defined as the constant entry and departure of nodes. Churn affects both structured and unstructured overlay networks and, in order to build resilient search protocols, it must be taken into account.

This dissertation proposes a novel search algorithm, called FASE, which combines a replication policy and a search space division technique to achieve low hop counts using a small number of messages, on unstructured overlays with non-hierarquical topologies. The problem of churn is mitigated by a distributed monitoring algorithm designed with FASE in mind.

Simulation results validate FASE efficiency when compared to other search algorithms for peer-to-peer networks. The evaluation of the distributed monitoring algorithm shows that it maintains FASE performance when subjected to churn.

KEYWORDS:
Peer-to-peer; unstructured overlays; churn; efficient search.

Resumo

Os sistemas *peer-to-peer*, como aplicações de partilha e distribuição de conteúdos ou voz-sobre-IP, são construídos sobre redes sobrepostas. Redes sobrepostas são redes virtuais que existem sobre uma rede subjacente, em que a topologia da rede sobreposta não tem de ter uma correspondência com a topologia da rede subjacente.

Ao contrário das suas congéneres estruturadas, as redes sobrepostas não-estruturadas não restringem a localização dos seus participantes, ou seja, não limitam a escolha de vizinhos de um dado nó, o que torna a sua manutenção mais simples. O baixo custo de manutenção das redes sobrepostas não-estruturadas torna estas especialmente adequadas para a construção de sistemas peer-to-peer capazes de tolerar o comportamento dinâmico dos seus participantes, uma vez que estas redes são permanentemente afectadas pela entrada e saída de nós na rede, um fénomeno conhecido como *churn*.

O algoritmo de pesquisa mais comum em redes sobrepostas não-estruturadas consiste em inundar a rede, o que origina uma grande quantidade de mensagens duplicadas por cada pesquisa. A escalabilidade destes algoritmos é limitada porque consomem demasiados recursos da rede em sistemas com muitos participantes. Para reduzir o número de mensagens, as redes sobrepostas não-estruturadas podem ser organizadas em topologias hierárquicas. Nestas topologias alguns nós da rede, chamados supernós, assumem um papel mais importante, responsabilizando-se pela localização de objectos. A utilização de supernós cria novos problemas, como a sua selecção e a dependência da rede de uma pequena percentagem dos nós.

Esta dissertação apresenta um novo algoritmo de pesquisa, chamado FASE, criado para operar sobre redes sobrepostas não estruturadas com topologias não-hierárquicas. Este algoritmo combina uma política de replicação com uma técnica de divisão do espaço de procura para resolver pesquisas ao alcançe de um número reduzido de saltos com o menor custo possível. Adicionalmente, o algoritmo procura nivelar a contribuição dos participantes, já que todos contribuem de uma forma semelhante para o desempenho da pesquisa. A estratégia seguida pelo algoritmo consiste em dividir tanto os nós da rede como as chaves dos seus conteúdos

por diferentes “frequências” e replicar chaves nas respectivas frequências, sem, no entanto, limitar a localização de um nó ou impor uma estrutura à rede ou mesmo aplicar uma definição rígida de chave. Com o objectivo de mitigar o problema do churn, é apresentado um algoritmo de monitorização distribuído para as réplicas originadas pelo FASE.

Os algoritmos propostos são avaliados através de simulações, que validam a eficiência do FASE quando comparado com outros algoritmos de pesquisa em redes sobrepostas não-estruturadas. É também demonstrado que o FASE mantém o seu desempenho em redes sob o efeito do churn quando combinado com o algoritmo de monitorização.

PALAVRAS-CHAVE:
Peer-to-peer; redes sobrepostas não-estruturadas; churn; pesquisa eficiente.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Overlay networks form virtual topologies on top of an underlying network. These topologies are said to be virtual because links in the overlay have no correspondence with links established in the underlying network. Peer-to-peer (P2P) networking is one of the applications that are based on overlays. Key applications of P2P include voice-over-IP (e.g. the Skype network) and software distribution (for instance, many Linux distributions are downloadable using P2P). More recently, the BBC made available its programs for download or streaming using P2P-based technology (BBC, 2008).

In a P2P network, resources are stored in a distributed fashion by the participating nodes. However, these resources need to be known to be of use to anyone. For this reason, P2P networks need to provide a search functionality to their users. Early P2P systems were built using a hybrid architecture, where some features like search depended on a centralized server, but, in recent years, P2P systems have moved from hybrid networks to fully decentralized architectures. Without a central directory, resource location becomes a problem because peers only have a partial view of the network. Besides resource location, P2P decentralized networks must also cope with the intermittent connectivity of the participants who may, intentionally or not, depart from the overlay at any time.

Current decentralized P2P search systems are built over structured or unstructured overlays. The former organize peers in a structure that strictly conforms to a desired topology (e.g. a ring or a torus) and that restricts which connections peers may establish as well as the resource placement. The unstructured variant offers few or no restrictions, allowing the peers to connect in a random topology,

but it may leverage on a hierarchical organization.

Locating an item in unstructured overlays with moderate resource consumption is an interesting challenge. Original resource location algorithms for these kind of overlays were reliable but resource intensive, what raised scalability problems, emphasized by the increased popularity of P2P networks in the Internet. The problem becomes more relevant when we known that participants abandon and join the overlay at a high rate (Liang *et al.*, 2005; Qiao & Bustamante, 2006; Saroiu *et al.*, 2002; Stutzbach & Rejaie, 2006), in a process designated as *churn*. Therefore, scalable search algorithms need to balance the consumption of network resources with the need for reliability.

The work presented in this dissertation is about resource location in unstructured overlays. The focus of the work is in search efficiency, that is, locating resources within a small number of hops using a relatively low number of messages, while considering the effect of churn.

1.1 Motivation and Contribution

Churn is an important characteristic of P2P file-sharing and content distribution systems. The negative effects of churn are more significant on overlays which put additional effort on maintaining a predefined topology. Operating under churn, structured distributed hash tables (DHT) experience an increase in lookup latency as well as inconsistent results (Rhea *et al.*, 2004). Additionally, DHT filiation can have a higher cost than filiation in unstructured overlays as, instead of joining a random neighbor, a node is restricted to join nodes that are close (the closeness is usually measured by the proximity of the node's IDs).

The search algorithms for unstructured overlays are often based in *flooding*, a reliable but resource intensive and unscalable method that is similar to a breadth-first search (with added redundancy), or *random-walks*, which use less resources than the previous method but cover less search space for each hop. The first generation of Gnutella (Clip2, 2001), a protocol where the impact of churn is minimal, uses a simple flooding search mechanism.

In unstructured hierarchical overlays, such as the current Gnutella protocol, a small subset of nodes are elected as *supernodes*. In such two tiered architectures, the contents offered by a regular node are indexed in the correspondent supernode

and queries are transmitted between supernodes. Supernode architectures exhibit an unfair load balancing between ordinary nodes and supernodes, since a supernode is not a dedicated server but a regular user similar to the other participants. However, a supernode can have a much higher load than regular nodes. Additionally, the loss of a supernode due to churn results in several nodes being orphaned, and their contents unreachable, until a new supernode is found. After a supernode is found, the nodes contents have to be indexed in their new supernode.

The present work is motivated by the observation that a fair network is also a more resilient one as, if all the nodes have the same role the loss of one can be more easily tolerated. Moreover, a "fair" organization might appeal to users who do not want to contribute with more resources than their peers. The challenge is how to achieve low latency searches and scalability in such networks, since flat, i.e. non-hierarchical, unstructured overlays that use flooding based search are not scalable. The introduction of supernodes, to overcome the scalability issue, results in an excessive unbalance as it imposes a much higher resource consumption to a few nodes than to the majority, and reduces the overlay robustness as it gives a more important role to a small subset of nodes. We note that even when supernodes have the capacity to cope with the additional load, the unfair load balancing may not be acceptable for some users.

The primary goal of this work is to design a search algorithm for unstructured overlay networks to improve fairness, while being able to find resources within a comparable number of hops with other search strategies. The algorithm should provide a robust and scalable alternative to Gnutella without requiring an underlying hierarchical organization of the participants.

The contributions of this dissertation are:

- the Frequency-Aware SEarch (FASE) algorithm, a search protocol over unstructured, non-hierarchical overlays.
- a distributed monitoring algorithm to help FASE cope with high churn rates.

FASE uses a search space partitioning mechanism, combined with a replication policy and biased random walks. The algorithm does not rely on a hierarchical architecture and does not require the exchange of routing information between

nodes, which would impact on its capability to recover from node failures. The search is an uninformed search that leverages on the partitioning mechanism. Additionally, we implemented a distributed monitoring algorithm that improves the performance of FASE while subjected to churn and removes references to outdated resources.

1.2 Organization

The remainder of this dissertation is organized as follows.

Chapter 2 presents the related work. Fundamental concepts about overlay topologies are defined and P2P networks participants and content distribution characterized. The participation characteristics in P2P networks are related to a model of churn that is representative of different P2P networks. Finally, a number of relevant P2P protocols are discussed.

Chapter 3 describes the Frequency-Aware Search search space division method, the replication and the search algorithms. The related monitoring algorithm is presented, followed by the implementation details of both algorithms. This chapter is concluded with a discussion of FASE.

Chapter 4 presents the evaluation of the algorithm implementation, based on extensive simulations. Additionally, FASE performance is compared with another search algorithm for unstructured overlays.

Chapter 5 concludes with the final remarks and discussion of future work.

Chapter 2

Related Work

There is a substantial body of research on P2P search algorithms, as well as numerous measurement studies about participant behavior and content distribution. In this chapter, we first refer to the topological aspects of P2P overlays before proceeding to characterize participant behavior and content distribution, which have to be considered when designing a search algorithm. Two classes of search algorithms for unstructured overlays are then discussed, followed by indexing strategies in decentralized overlays. We conclude with a survey of existing search protocols for unstructured overlays.

2.1 Overlay Topologies

The topological characteristics of P2P overlays need to be taken into account when designing search algorithms. Overlay networks can be represented by a graph, usually a random graph in the case of unstructured overlays. In the case of structured overlays, they obey to well known topologies such as ring, torus, tree or a combination of the above. First, we define a few fundamental concepts that are essential to the discussion of network topologies:

Degree In an undirected graph the degree is the number of edges of a node. In a directed graph the incoming edges of a node define its *in-degree*, whereas the outgoing edges define its *out-degree*.

Neighbor A node n has a neighbor n' if there is an edge connecting n and n'. The set of neighbors of a node defines the nodes *view*.

Filiation in unstructured overlays is usually straightforward: the node wishing to join the overlay first contacts a well known contact point or host cache[1] and either adds the contact point as its neighbor or obtains a list of nodes from the contact point from where a suitable neighbor is found. The node keeps looking for other nodes willing to accept connections until a defined limit is reached. To find these nodes, the node relies on explicit look-up messages or on the examination of the overlay traffic.

Unstructured overlays form random topologies with a diverse degree distribution. Measurement studies such as (Saroiu *et al.*, 2002) or more recently, (Stutzbach *et al.*, 2005) proved the robustness of such topologies: (Saroiu *et al.*, 2002) concludes that, for a maximum degree of 20, the Gnutella overlay is fragmented when more than 60% of the nodes fail but that it is vulnerable to the failure of 4% of the nodes with the highest degree, which suggests that, for a overlay where the median degree is low, a topology where all nodes have an approximate degree would be more reliable; (Stutzbach *et al.*, 2005) states that upon removal of 85% of the nodes the remaining 90% are still connected and, as a result of a higher median degree when compared to previous studies, the overlay is more resilient to the removal of high degree nodes.

In this work we considered two kinds of topologies: *i*) a Gnutella-like random graph where the node degree is randomly distributed; and *ii*) a random graph where nodes seek to maintain a constant degree. In all cases we consider undirected graphs. The constant degree topology is based on HyParView (Leitão *et al.*, 2007). HyParView is a robust membership protocol which tolerates massive failures up to 70% of the nodes before the overlay is fragmented.

2.2 Peer-to-Peer Networks Characterization

Characterizing user participation in P2P networks is a necessary step to establish a realistic model to be used in the evaluation of the impact of churn in P2P protocols. Churn, the continuous arrival and departure of nodes, is a direct consequence of P2P participants behavior. It is also necessary to characterize the content

[1]The existence of a well known contact point is a common assumption in overlay membership algorithms.

distribution, as the effectiveness of a search algorithm can be affected by the existence of content replication.

2.2.1 Participation Dynamics

The patterns of peer participation are discussed in several studies about widely used hybrid, structured and unstructured P2P networks (Liang *et al.*, 2005; Qiao & Bustamante, 2006; Saroiu *et al.*, 2002; Stutzbach & Rejaie, 2006), where similarities in peers *session length* in different P2P networks have been found. The session length metric is frequently used in the literature to characterize churn in P2P networks. It is defined as the time between the arrival and departure of a node. (Stutzbach & Rejaie, 2006) surveyed three types of deployed P2P networks by observing three popular P2P services, Gnutella, BitTorrent and Kad. The current generation of Gnutella implementations are based on a supernode architecture; BitTorrent is a content distribution system where peers form a distinct overlay for each file and locate each other via a well known point called a tracker; and Kad is a P2P network based on the DHT Kademlia (Maymounkov & Mazières, 2002).

In (Stutzbach & Rejaie, 2006), Weibull[2] and log-normal distributions were identified as the more accurate distributions for modeling session lengths. In the majority of the studies cited above, it was observed that:

- The session length distribution across the three different systems is similar.
- The session lengths observed in different systems can be described by the same type of distribution.
- Within the same system, the distribution of session length for different data sets is similar.

Figure 2.1 shows the plot of the session length distribution for three distinct BitTorrent data sets, using the data in (Stutzbach & Rejaie, 2006). The plot shows the similarity of the session lengths in the different data sets, all described by a Weibull distribution.

[2]The Weibull distribution is a continuous probability distribution commonly used in life data analysis (for instance, to predict the failure of machine parts). Its probability density function is given by $f(x) = \frac{k}{\lambda}(\frac{x}{\lambda})^{k-1}e^{-(x/\lambda)^k}$.

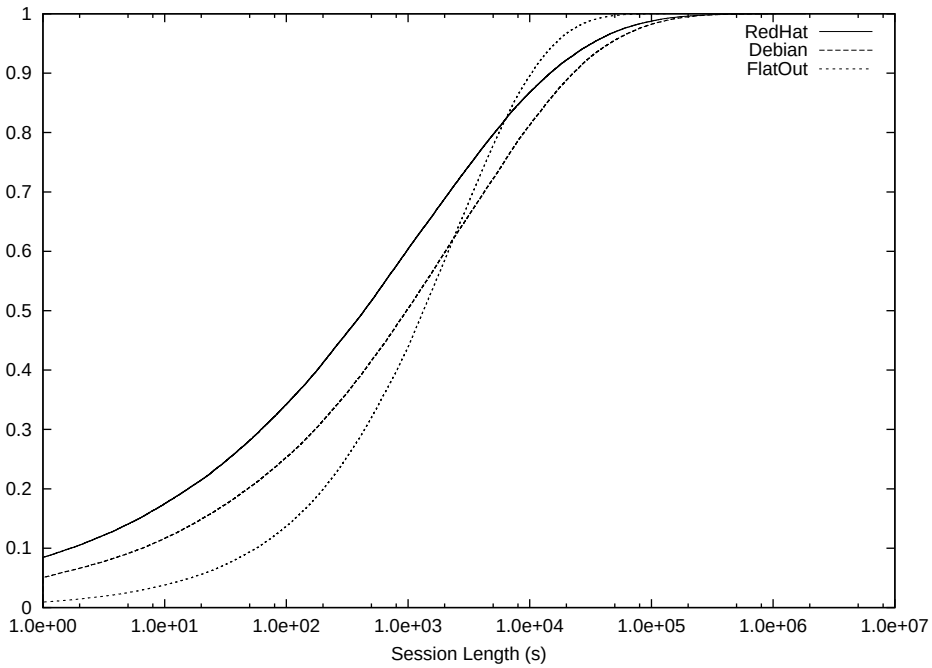


Figure 2.1: Session length distribution of three data sets within the same system.

The consistent observation of similarities in different studies, focused on different P2P networks, suggest that user behavior does not differ from network to network. Consequently, the churn rate is similar across different networks and a common model for user participation can be used when evaluating different P2P networks.

2.2.2 Content Distribution

The characterization of content distribution is important because some algorithms, such as Gnutella, are more suited to find well replicated content whereas others, for instance a DHT (if not impaired by the effect of churn), are efficient at finding even the rarest of objects, since some algorithms use blind search methods and others know the precise location of items.

The contents available in P2P networks are not homogeneous, persistent or evenly distributed. The transiency of P2P peers discussed in the previous section in part justifies this as, if the participants who offer the content are highly transient, the content offered by those participants will likely be affected by that

behavior. Furthermore, most P2P content sharing systems do not force participants to share content (at least at the protocol level). It has been noticed that many participants do not share any resources. Such participants are known as *free-riders*. For instance, (Saroiu *et al.*, 2002) found that 25% of Gnutella peers do not share any files; (Fessant *et al.*, 2004) found 68% free-riders in the eDonkey networks.

Survey studies found that the content distribution is similar to the query distribution at a given moment, that is, how well an item is replicated is closely related to its popularity. (Gummadi *et al.*, 2003) and (Fessant *et al.*, 2004) findings show that *i*) the popularity of the available content follows the same pattern whether measured as item replication or as the number of requests for items; *ii*) the most popular items are recently born; *iii*) highly replicated items have a uniform distribution and the least popular items follow a Zipf distribution. Zipf law states that the frequency of an item with rank n is proportional to $\frac{1}{n^{\alpha}}$, where α is close to 1.

2.3 Search Algorithms

In this section we discuss with more detail the flooding and random walks algorithms introduced in the previous chapter.

2.3.1 Flooding

In a flooding search each node forwards a message to every neighbor, except to the node from where the message was received. Propagation of messages is limited by a time-to-live (TTL) field that is decreased every hop and, additionally, nodes do not forward messages seen before. When a message is propagated by flooding it is guaranteed that it is delivered to every node that can be reached within TTL-hops, however, there is a high number of redundant messages, since every node retransmit the message upon seeing it for the first time. Therefore, flooding searches are reliable but result in a high overhead, particularly in high degree graphs. Moreover, in (Lv *et al.*, 2002) it is shown that the percentage of redundant messages increases with the TTL.

The definition of a correct TTL is itself a problem as, if it is set too high it implies a large overhead, but if it is too low the coverage is reduced and a node that uses flooding can only know the contents of the nodes at a distance (in hops)

of the designated TTL.

There are two variants of flooding which present trade-offs between reliability, search latency and message overhead. One option to mitigate flooding overhead is an expanding ring, or iterative deepening, search (Lv *et al.*, 2002; Yang & Garcia-Molina, 2002). The search starts with a small TTL; if it is not successful the TTL is increased and the search is started again. This method can only be efficient if the requested item is close to the source. If the requested item is not close to the query source, it may consume more messages than the standard flooding method in consequence of the successive iterations. Another approach is to only forward a query to a subset of the neighbors of a node according to some deterministic metric (Yang & Garcia-Molina, 2002) (e.g. quality of past results) or with some probability p. While this method reduces the overall number of redundant messages, it also reduces the coverage of a search and increases latency. However, this method also loses reliability. If the loss of reliability is acceptable, a search algorithm that does not generate redundant messages can be more adequate.

As a consequence of its message redundancy, flooding is not feasible for large scale networks. Even in smaller networks the selection of a correct TTL poses a non trivial problem.

2.3.2 Random Walks

In random walks, messages are not replicated. Instead, when receiving a message for the first time, each node forwards it to exactly one of its neighbors selected at random. It was shown that random walks can find objects within a comparable number of hops, while reducing network traffic by an order of magnitude when compared to flooding (Lv *et al.*, 2002).

A single random walk may traverse a large number of hops, adding latency to the search. The performance of random walks can be improved if the selection of the next hop is biased to nodes where the location of the item is more probable. We defer the specifics of this biasing to the discussion of existing protocols, which use random walks to achieve scalability, in Sec. 2.5. Another method to improve the performance of random walks is the use of multiple instances or *walkers*. However, this method evinces the same problem of termination that characterizes a flooding algorithm. Unlike a flooding algorithm, the number of messages remains constant during the search (or, equal to the number of walkers); unlike flooding, it is feasible

to contact the query source to check for termination. If such a method was to be used in a flooding there would be a message implosion at the source node, as the number of messages grow by each visited node. An adaptive termination algorithm that uses this method is described in (Lv *et al.*, 2002). In this algorithm, each n hops, the walkers contact their origin to check for termination before proceeding to the next hop. A TTL, set to a large value, is used to prevent loops.

In a network with high clustering, that is, when nodes have many neighbors in common, the probability of finding content outside clusters diminishes, since a walker can exceed its TTL without ever leaving a cluster. Cluster Resilient Random Walk (Chen *et al.*, 2007) was proposed to mitigate this problem. In this algorithm, the probability of a walker being forwarded is higher for neighbors that have fewer nodes in common. The drawback of this algorithm is that it imposes a non-negligible cost, since every node has to know the view of its neighbors, which may change frequently.

2.4 Indexing

Any search method requires the existence of some index to reference the contents that are to be the subject of search. In decentralized P2P networks we distinguish between local and distributed indexes.

Local Indexing Local indexing is characteristic of the first version of the Gnutella protocol, where each node only indexed its local content. This indexing scheme has the advantage of allowing unrestricted search as there is no loss of meta-information because the content is local. However, this method limits scalability inasmuch as a query for an item has to visit the exact nodes which have the item. To the best of our knowledge, every currently deployed P2P network use a distributed index.

Distributed Indexing In this scheme the contents of a node are indexed in multiple nodes. Distributed indexes are implemented in a number of ways. Protocols that are based on two-tiered hierarchical overlays index the information of every regular node in the corresponding supernode. One technique called one-hop replication dictates that pointers to the items stored in node n are indexed in every neighbor at the distance of one hop of n. Freenet (Clarke *et al.*, 2001) distributes items instead of references to items. Distributed indexes can be used to support

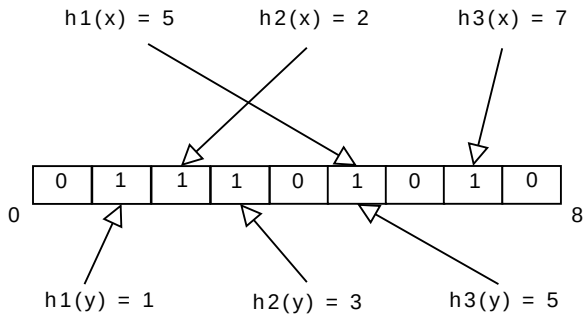


Figure 2.2: Insertion of elements in a Bloom filter.

routing decisions in search algorithms.

In the following section we describe a data structure that can be used to save and transmit indexes in a space-efficient way.

2.4.1 Bloom Filters

A Bloom filter (Bloom, 1970) is a lossy data structure used to test membership. It is constituted by a bit vector V of length m, initially with all the elements set to 0. A Bloom filter uses k hash functions. Upon insertion, an item i is hashed k times, one for each different hash function. The result of applying the k hash functions is that k bits are set to 1 (assuming that there are no collisions). Figure 2.2 depicts the insertion of two elements, x and y in a Bloom filter with $k = 3$. When a bit is already set to 1, as is in the case of the 5Th bit of the filter, shared by $h1(x)$ and h3(y), the value remains unchanged.

To test if some item is included in the filter, the same k hash functions are applied. An item is assumed to be included in the filter if all the bits returned by the k hash functions are set. Given that a single Bloom filter may include many items, a test may return a false positive. Assuming perfectly random hash functions, the probability of obtaining false positives is given by $(1 - (1 - \frac{1}{m})^{nk})^k$, where n is the number of elements in the filter (Broder & Mitzenmacher, 2004).

Bloom filters have traditionally been used in database applications, for instance, BigTable (Chang *et al.*, 2006) uses Bloom filters to reduce the number of disk accesses. More recently, they have been used for network applications, which include the construction of compact indexes or probabilistic routing tables on P2P

overlays (Broder & Mitzenmacher, 2004).

2.5 Search Protocols in Unstructured Overlays

In this section we discuss various existing protocols for search in unstructured overlays. The protocols discussed here are representative of the various approaches to search in unstructured overlays.

2.5.1 Gnutella

The original Gnutella (Clip2, 2001) used flooding over a flat unstructured overlay. A node wishing to join the overlay first discovers other peers using a host cache. Further neighbors are discovered using PING messages, which are answered with multiple PONG messages. A node may send PONG messages with its own address or with cached addresses from previously seen messages. Search is performed using QUERY messages which are answered with QUERY_HIT messages, that are sent via the query reverse path. Gnutella ping and search messages use the flooding algorithm described in Sec. 2.3. Therefore, Gnutella is reliable, and is not impaired by churn, but suffers from the scalability and TTL limit problem of a pure flooding search.

2.5.2 Hierarchical Overlays: Kazaa

Kazaa (Liang *et al.*, 2005) uses a two-tiered hierarchical architecture similar to the current generation of Gnutella. Two-tiered architectures were developed in response to Gnutella scalability issues. Kazaa distinguishes between Ordinary Nodes (ON) and Super Nodes (SN). Each ON connects to a SN and SNs establish connection between each other. Queries, which are keyword-based, are directed to the corresponding SN n and possibly forwarded to a subset of the SNs known by n. In Kazaa, neighbor selection is affected by locality, which reduces latency but reduces the scope of the results.

A SN contribution is much higher than the contribution of ONs: each SN keeps about 100 to 200 connections to ONs and 40 to 60 to other SNs. Additionally, a SN has to index all its ONs contents and maintain that index updated to avoid false positive search results. Furthermore, the overlay depends on a small subset of nodes, that are not dedicated servers like in some centralized services, but ordinary

peers. Consequently, the Kazaa overlay, and this type of architecture in general, lacks load balancing and its robustness could be improved.

2.5.3 Semantic Communities

A class of protocols establish semantic or interest based relationships between nodes to improve search. *Acquaintances* (Cholvi *et al.*, 2004) adapts the overlay topology to establish communities of peers with the same interests. Acquaintances nodes keep *neighbor links*, chosen randomly like in Gnutella, and *acquaintance links* to nodes which returned a positive response to a previous query. Search is performed using flooding with a low TTL, as acquaintance links should place the desired content at a shorter distance than in the standard Gnutella overlay.

A rationale similar to the one followed by Acquaintances has been proposed to work on top of the Gnutella overlay (Sripanidkulchai *et al.*, 2003). In this algorithm, a peer finds shared interests in peers that have similar contents, and establishes shortcuts to those peers. The similar contents are found at first by flooding and, subsequently, from query results and by observing the network traffic. Queries are directed first to the shortcut links and, upon failing, queries use the Gnutella flooding scheme.

The *Self-Organizing Super-Peer Network* (SOSPNet) (Garbacki *et al.*, 2007) is a supernode architecture that explores the common interest of the participants. Unlike other P2P networks, where regular nodes are connected to one supernode and the later is responsible to index the regular nodes contents, in SOSPNet the contents of a regular node may be indexed across several supernodes. Figure 2.3 depicts SOSPNet architecture:

- Each regular node (designated as a *weak peer* in SOSPNet) maintains a cache of supernodes (designated as *super-peers* in SOSPNet). The cache is ordered according to the supernode's priority, which is increased when a supernode returns a positive reply to a query. When the cache capacity reaches its limit, the item with the lowest priority is discarded.

- Each supernode maintains a cache of pointers, organized by priority, that reference items located in regular nodes. The cache is updated when a supernode receives search results. If the pointer already exists, its priority is

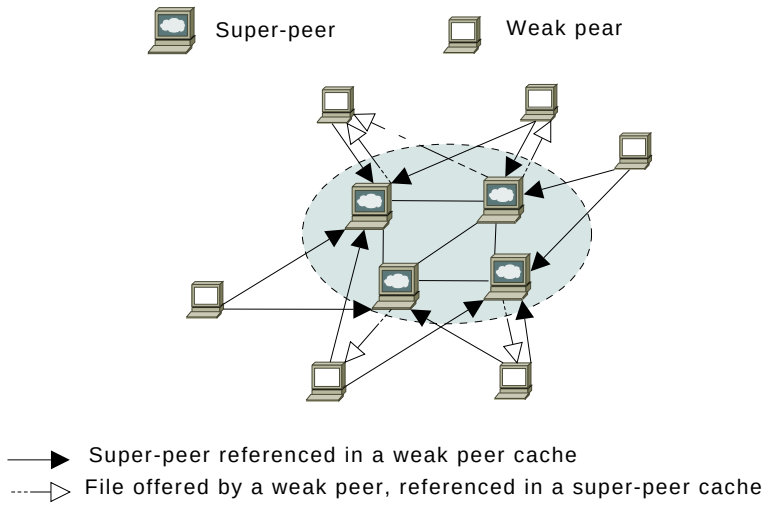


Figure 2.3: SOSPNet architecture.

incremented; a pointer to a new resource is given the value of the maximum priority + 1.

Periodically, the regular nodes send information about their resources to a random supernode, where the probability to be chosen is proportional to the supernode priority in the cache maintained by the regular node. Furthermore, when a search succeeds, the node that initiated the search merges into its cache the supernodes present in the cache of the node which replied to the search. Therefore, the regular nodes establish interest relations with the supernodes that satisfy their requests, as supernodes with a higher priority are the ones that satisfied more requests from the regular node. It is assumed that, if two nodes have similar interests, they will search for similar content. Therefore, their search is optimized to the same supernode, as the supernode that satisfies both nodes requests will have a high priority in their caches. The search protocol operates in the following manner:

- A regular node starts the search in the supernodes it knows, according to their priority. If the item is found, the corresponding supernode priority is updated.
- When an item cannot be found in the way above, the query is forwarded to one of the supernodes, preferably with high priority. The supernode will be

responsible to find the item using its connection to the other supernodes.

- Overloaded supernodes reject requests, thus diminishing their priority and the probability of receiving more queries.

This algorithm is interesting as regular nodes do not need to rely on a single supernode as in other architectures, but the interest relations, in the form of the weak peers and super-peers caches, require previous searches to be effective. The search will be similar to the traditional supernodes architecture for new nodes entering the overlay.

Algorithms that identify relations between nodes interests are feasible in stable networks where peers have long session times and perform many queries for the same type of content. Maintaining interest relationships in a network where most of the nodes have short session times is more complex, as it is unlikely that a nodes performs or answers to more than a few queries. The fundamental flaw of this class of algorithms is to assume that the relations hold or can remain reliable when the overlay is subject to churn.

2.5.4 Adaptive Probabilistic Search

Adaptive Probabilistic Search (APS) (Tsoumakos & Roussopoulos, 2003) uses biased random walks. Each node has an index with an entry for each object referred in a query it has forwarded, and a value for the link to where the query was forwarded. Values can be updated according to two policies: *i*) an optimistic policy that assumes that the query will have success and dictates that the value of a link is increased after a query is forwarded through that link; or, *ii*) a pessimistic policy, where the value is decreased because it is assumed that the query will fail. If the queries do not yield the expected result the values are corrected using an update message propagated using the reverse path. There is no recovery mechanism if the path is broken (for instance, by a node departure).

Unfortunately, this method has limited applicability when the probability of having a large number of queries to the same item hitting the same node is low. Furthermore, the information collected from past searches is quickly outdated, as the arrival and departure of nodes breaks the paths traversed by previous queries.

2.5.5 Gia

Gia (Chawathe *et al.*, 2003) was designed considering the heterogeneity of P2P systems, documented in (Saroiu *et al.*, 2002). It adapts the overlay topology to the capacity of the participating nodes and directs queries to the higher capacity nodes. Furthermore, it defines a flow control policy where nodes grant flow control tokens according to their capacity, and a replication policy that dictates that the index of a node is replicated on its neighbors at the distance of one-hop.

The topology adaptation algorithm depends on the following parameters:

Capacity The capacity of a node is a system parameter defined, for instance, by the node bandwidth and/or CPU.

Minimum Number of Neighbors The topology adaptation algorithm searches for new neighbors at least until the defined minimum is reached.

Maximum Number of Neighbors This parameter bounds the maximum number of neighbors that a node may have. Each node has a local maximum computed from this parameter and the nodes capacity. The local maximum can be lower but not higher than this parameter.

Satisfaction Level Gia computes a "satisfaction level" S, $0 \leq S \leq 1$, for each node. The satisfaction level at each node depends on the node's neighbors. The contribution of a neighbor on the satisfaction level is defined by the ratio of the neighbor capacity to the neighbor degree.

Nodes perform topology adaptation until $S = 1$, even if some neighbors have to be dropped to be substituted by other nodes which provide a higher satisfaction. A node accepts new connections until the maximum number of neighbors is reached and may drop existing neighbors if the new connection offers better capacity.

Queries are only forwarded to nodes with available flow control tokens. Each query is a single random walk. Gia preferably routes random walks to the nodes with higher capacity, those that are more likely to be more connected, thus more likely to know an item due to the one-hop replication policy.

The result of the topology adaptation algorithm is a hierarchical overlay, however distinct from the two-tiered supernode approach. Queries use a single random walk which requires the use of "keep-alive" messages which, like query responses, are sent via reverse path. Such paths will be frequently broken by the topology adaptation algorithm but, when subjected to churn, they will also be broken by the arrival and departure of nodes. Gia addresses this problem by keeping connections open to nodes dropped as a result of a topology adaptation decision, but this does not prevent the loss of queries due to churn.

2.5.6 Scalable Query Routing

Scalable Query Routing (SQR) (Kumar *et al.*, 2005) uses information encoded in probabilistic routing tables to forward queries to the (probable) location of the desired object.

SQR routing tables store one Exponentially Decaying Bloom Filter (EDBF) for each neighbor link as well as a local EDBF. The local filter is an index of the node contents. An EDBF is similar to the standard Bloom filter described in Sec. 2.4.1, but the membership of an item in a Bloom filter is tested by a function $\theta(x)$ which yields the number of bits in the filter that are set to 1.

SQR periodically transmits routing table updates. When creating an update, SQR decays filter information according to a parameter d. To create an update from node n to node n', n first merges all its neighbors corresponding filters, with the exception of the filter from n'. The resulting filter is decayed, that is, its bits remain set to 1 with a probability $\frac{1}{d}$. The decayed filter is merged with n's own (undecayed) local filter; finally, the resulting filter is forwarded to n'. Figure 2.4 shows the result of the decay operation: node B knows $\frac{1}{d}$ of the information of the other neighbors of A; node C knows $\frac{1}{d^2}$ of the information of A's neighbors and $\frac{1}{d}$ of the information of the other neighbors of B.

Queries for a key x are forwarded to the closest neighbor n, that is, when the routing table associated with a link to n has the greater $\theta(x)$. When a query revisits a neighbor it is randomly forwarded as it is assumed that the previous routing decision was wrong and therefore the routing tables are invalid.

The maintenance of the probabilistic routing table implies a non-negligible message overhead. To mitigate this problem, information for each update degrades with the distance to the host, thus implying an increasing "noise". In consequence,

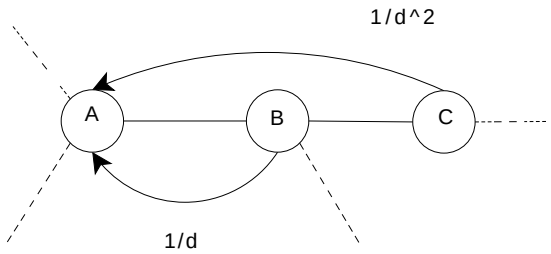


Figure 2.4: Decaying information in a Exponentially Decaying Bloom filter.

a search too far away from the searched object behaves like a random walk (and SQRs queries use only one random walk). The problem is amplified in the presence of churn, where the loss of accuracy caused by out of date tables will be the rule rather than the exception. To the best of our knowledge there is no study on the behavior of SQR under churn.

2.6 Summary

In this chapter, P2P overlays were characterized at topological and user level. Both should be considered when designing a search algorithm. The flooding and random-walk search algorithms were discussed, followed by the classification indexes (local and distributed) available in unstructured overlays. The search algorithms and the random-walks are closely related, as the indexing scheme may improve or impair query resolution. Several protocols were discussed: a flooding search protocol over flat unstructured overlays (Clip2, 2001) and a protocol for hierarchical overlays (Liang *et al.*, 2005); protocols which establish a relation between nodes based on data from past searches (Cholvi *et al.*, 2004; Sripanidkulchai *et al.*, 2003); a protocol that uses information from past searches to probabilistically route random walks (Tsoumakos & Roussopoulos, 2003); a protocol that combines topology adaptation and a replication policy to create clusters were the probability of finding an item is higher; and, a protocol which explicitly transmits and stores routing information used to forward queries (Kumar *et al.*, 2005).

Chapter 3

Search In Unstructured Overlays

Unstructured non-hierarchical overlays are attractive because they impose few constraints to the participating peers and have a low maintenance overhead. A well known example of this kind of overlays, the first implementation of Gnutella, had a simple and reliable architecture but could not scale to a large number of peers, since its flooding-based search algorithm resulted in an insurmountable overhead for every search. As a consequence of their scalability issues, Gnutella and similar architectures evolved to hierarchical overlays that organize themselves in regular nodes and supernodes. The later are a small subset of the overlay and the location of content depends entirely on them. The introduction of supernodes results in an excessive unbalance as it imposes a much higher resource consumption to a few nodes (see Sec. 2.5.2) than to the majority, and reduces the overlay robustness as it gives a more important role to a small subset of nodes.

This chapter describes an algorithm that leverages on the simplicity and low cost of non-hierarchical overlays, with the purpose of providing a scalable, efficient and robust alternative to Gnutella while evenly distributing the load between participants and maintaining a moderate cost of search and maintenance.

3.1 Frequency Aware Search

The *Frequency Aware SEarch*, or FASE, is a search algorithm for flat unstructured overlays. FASE objective is to achieve efficiency as well as to be more scalable when compared to flooding search algorithms. Furthermore, these objectives should be achieved with a fair distribution of the load for all the participants.

FASE defines a policy for partitioning the search space and a key replication

policy. It partitions both the key space and the nodes in a common "frequency space"[1].

A frequency is defined to be any element of a finite and discrete set Φ. We define functions $f_N : N \mapsto \Phi$ and $f_K : K \mapsto \Phi$ mapping respectively the set of nodes (N) and keys (K) in frequencies. Mappings should uniformly distribute nodes and keys by frequencies. A non-uniform mapping will result in unbalanced partitions leading to a sub-optimal performance of the algorithm. A good example of a mapping function is the hash of a node IP address and port. The set Φ and functions f_N and f_K are known by every node in the network. Node neighbors are freely created and are expected to be from different frequencies. A "frequency path" is defined as an uninterrupted sequence of neighboring nodes of the same frequency.

A *pointer* is a tuple defined as follows:

Pointer: ⟨Key,URL⟩

In a pointer, a key is some entity that can be used for identifying the item and an URL is an unique, complete address to a node. We defer a more precise definition of a key to Sec. 3.3.1. A pointer maps a key to the unique identifier of the node that is storing the item referred by the key. In FASE, pointers to the items are stored on nodes with the same frequency of the item's key. The placement of pointers is restricted to the same frequency of the key but the placement of the original item is not constrained in any way. FASE improves search results by having nodes to forward random walks preferably to nodes in the same frequency of the search key.

3.1.1 Assumptions

FASE makes no assumptions about overlay structure, nor is overlay membership the focus of this work. The algorithm does not require node membership to be constrained by the node identification or that node insertion conforms to a given structure. However, it is assumed that the majority of nodes (more than 95%) in the overlay have a constant degree. There are two reasons for this assumption.

[1] FASE was designed with wired peer-to-peer networks in mind. However, FASE search algorithm has some resemblances with a model where queries are broadcasted in a frequency echoed only by some objects.

First, with a uniform distribution of frequencies (assumed to be provided by functions f_N and f_K) and for a given degree and number of frequencies, each node will have the same probability of having a neighbor with some frequency f, that is, a node in f can be reached from every node n, $f_N(n) \neq f$, with the same probability p. Second, a constant degree, that can be offered by membership protocols such as Hyparview (Leitão *et al.*, 2007), evenly balances the load between nodes when combined with a uniform distribution of nodes and items over Φ.

3.1.2 Pointer Replication

To facilitate item location, nodes replicate pointers using REPLICATION messages. Each REPLICATION message carries one or more pointers with keys of the same frequency. Therefore, it is said that the REPLICATION message belongs to that frequency. For simplicity, we denote by $f_K(M_R)$ the frequency of the keys carried in message M_R.

REPLICATION messages carry a counter that defines the maximum number of replicas of each pointer to be stored, defined by a configuration parameter. The replica counter is decremented by every peer deciding to store a pointer and the message is discarded when the counter reaches 0. A REPLICATION message M_R is successively forwarded at each node $n \in N$ to one of n's neighbors (nn) not visited by the message. Preference is given to neighbors in the frequency of the message, that is, with $f_N(nn) = f_K(M_R)$.

A node n' stores the pointer when it belongs to the same frequency of the message and it has received a message from a node n with a different frequency, that is, $f_N(n') = f_K(M_R) \wedge f_N(n) \neq f_N(n')$.

Recall that the REPLICATION message is preferably forwarded to nodes labeled with the same frequency of the item. Therefore, if there is a path of nodes in the same frequency, only one pointer is stored for the whole path. The advantage in storing pointers in adjacent nodes is small, as searches are preferably forwarded to nodes in the same frequency. It is preferable to make pointers available in the highest possible number of paths. The goal of this replication method is to make items well available for searches starting from any node in the overlay.

The pointer replication algorithm is illustrated in Fig. 3.1, which shows the path of a REPLICATION message M_R starting at node S. The interrupted circles denote nodes in the same frequency of the message. Node S does not have any

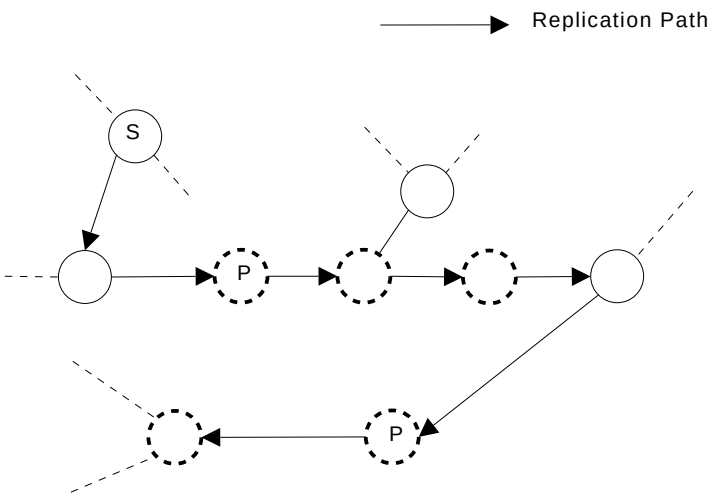


Figure 3.1: Pointer replication.

neighbor in M_R frequency, $f_K(M_R)$, hence the message is forwarded to a random neighbor; in the following hop there is a neighbor n with $f_N(n) = f_K(M_R)$ and subsequently the message is forwarded to that neighbor, which will store a pointer. The node that stores a pointer has a neighbor of the same frequency to where the message is forwarded, but that will not store the pointer since it is in the same frequency of the previous node. The figure shows that a REPLICATION message is always forwarded to (unvisited) nodes of the same frequency. The nodes labeled with P store the pointer carried by M_R. Note that the pointers are stored once per path. The forwarding of M_R stops when the replica counter reaches 0 or when all the neighbors of the current node have been visited before by M_R.

While the process of replication may sound costly, as several nodes may be visited before the replica counters reaches 0, keep in mind that it happens only once for a key or set of keys. A space efficient method to group keys with the same frequency and bound the number of replication messages per node to the number of frequencies will be described in Sec. 3.3.

3.1.3 Item Retrieval

If the mapping functions, f_N and f_K, uniformly distribute nodes and keys by frequencies, the search space of some key should be approximately $\frac{|N|}{|\Phi|}$ in comparison with $|N|$ of the original Gnutella and similar algorithms. Depending on the number of frequencies $|\Phi|$, this may represent a substantial reduction of the search space.

The retrieval algorithm takes advantage of this property by forwarding random walks preferably to nodes in the same frequency of the key.

To locate an item (through the location of the keys contained in the disseminated pointers), peers use QUERY messages, forwarded in one or more random walks. The message includes a search string that is mappable to the same frequency of the key. Queries are forwarded to a neighbor picked randomly from the set of unvisited neighbors with the same frequency of the key; or, if there is no node with the same frequency, to a random unvisited neighbor. That is, once a random walk enters a path of nodes in the frequency of the target key, it will follow that path. Therefore, if a pointer to the item was stored in the path, the random walk should find it before exiting the path. Success depends on the node where the random walk enters the path and the direction taken by the message when following the path.

When a QUERY message is delivered to a node, the node first verifies the items that are stored at the node. If the node has the same frequency of the query, then the pointers stored in the node are verified. Upon a hit, i.e. when an item corresponding to the query criteria is found, the search terminates and the associated URL is returned.

3.2 Monitoring Algorithm

The monitoring algorithm main contribution is to prevent query performance degradation due to the progressive loss of replicated pointers resulting from churn. As it is closely related to FASE, henceforth we will refer to this algorithm as FASE+. FASE+ aims at maintaining a sufficient number of pointers. Without pointers, items may still be reachable, but only within an abnormally high hop count, as the random walks will be biased to nodes where the item cannot be found (unless the key frequency is coincident with the frequency of the node storing the item).

A possible approach to mitigate the progressive reduction of the number of pointers would be to periodically trigger a new replication phase. However, a fundamental aspect of FASE is that the replication algorithm stores a limited number of pointers. Moreover, a pointer is stored only when the replication message enters a path (of nodes in the same frequency) and, due to the topological changes

resulting from churn, one can not expect to be able to replay the path followed by a previous REPLICATION message. The periodic repetition of the dissemination algorithm, without knowledge of the path followed by a previous edition, could generate an unaccountable number of pointers as some from the previous dissemination would still exist. Furthermore, with this method, it would not be possible to determine which items had lost more pointers. Consequently, in the worst case there would be an excessive replication of pointers i.e., the total number of pointers would increase over time, thus wasting bandwidth and memory at the nodes.

3.2.1 Backup Creation

A strategy to avoid the loss of pointers based on their repeated dissemination could present a high overhead and would be inaccurate. A better approach is to monitor the state of the existing pointers. FASE+ follows a decentralized monitoring approach, storing backup pointers. Each *backup* is closely associated to a single pointer and is preferably stored in the same path. FASE+ deploys at most one backup per pointer on the channel. In FASE+, backups replace the primary pointers when the later are lost due to the departure of nodes from the overlay.

When a node, lets call it N_p, is requested to store a pointer of some item, it immediately creates and forwards a backup. The backup is forwarded to a neighbor of N_p in the same frequency. If there is not any neighbor in the same frequency, the first neighbor is chosen from the node's ordered list of neighbors. When the backup message reaches a neighbor in the same frequency that is not holding a pointer of the item, the backup is stored and the message is discarded. The node storing the backup associates with it the identity of N_p. When a node stores a backup, it sends back to the backup source (that is, N_p) a message to register itself. In this manner, nodes holding pointers and their respective backups monitor themselves. Therefore, there is, at most, one backup per pointer.

Let Δ be the timeout for backup verification, N_p a node holding the pointer p and N_b the node holding b, that is the backup of p. Primary and backup nodes periodically cross-check each other according to the following rules:

- Every Δ, counting from the moment when the node joined the overlay, N_b will try to contact N_p. If N_p has failed, b is self promoted to a primary pointer p' and a new backup b' is created using the procedure described above.

- Every Δ, N_p will try to contact N_b. If N_b has failed, N_p creates a new backup using the procedure described above.

FASE+ backup deployment algorithm exhibits two interesting properties: *i*) with a high probability, FASE+ will deploy one backup per path, thus contributing to a good pointer distribution, even in the presence of churn; and *ii*) backup forwarding is a low overhead operation because in most cases, some one-hop neighbor of N_p will satisfy the conditions required for becoming a backup holder.

3.2.2 Stale Items Update

In an overlay subject to churn, pointers to items which are no longer available may persist. FASE+ uses two strategies to deal with this problem. For simplicity, we designate the node referred by a pointer as the *holder*, that is, a node that holds some referred item. First, before creating a backup from a pointer p, the node that has p tries to contact the holder. If the holder is down, p is removed and the backup creation is aborted.

The second strategy is implemented in the search algorithm. Whenever a query finds a pointer which matches its search criterion, it contacts the holder. If the holder is unreachable or does no contain the item, the pointer is removed and the query continues. In this scenario, the backup of the pointer that is removed would still be out of date. For this reason, when the loss of a pointer is detected and a backup is to be promoted, the node tries to contact the holder and, upon verifying that the holder is down, removes the backup.

3.3 Implementation

In this section, we discuss the implementation options of the algorithms presented in the previous sections. There are a number of options to be weighed such as the definition of key, which can limit what is searchable in FASE; the implementation of random walks and related configuration parameters, that can have an impact on FASE performance; and how to reduce the cost of replication when multiple keys have to be disseminated.

3.3.1 Key Definition

FASE does not enforce a strict definition of key. Instead, it can be adapted to the preferred type of search. Two possible definitions of key are:

- a **String**, one or more words describing the item;
- a **ID**, the unique id identifying an item, e.g. a cryptographic hash of the item.

FASE only requirement for a key is that it must be mappable to a frequency. To broaden the possible search criteria, keys should be as flexible as possible. We note that a key does not need to identify an unique item. For instance, if it is a **String**, it can be a single word that is mappable to several items. Following the same rationale, an item may be associated to different keys, for instance, based on a set of strings that describe the item.

3.3.2 Dissemination of Multiple Keys

In real scenarios peers usually make available a (possibly large) number of data items. A single Replication message can be used to announce multiple items and/or multiple keys of the same item, if we extend the syntax of the pointer construct. Since the keys may share a common frequency, a pointer can carry multiple keys of the same frequency. In FASE, the pointers transmitted in Replication messages can be defined as a tuple ⟨**Set**⟨**Key**⟩**,URL**⟩, that refers multiple keys sharing the same frequency.

The key set can be represented in a space-efficient way by using Bloom filters. To advertise multiple keys belonging to different frequencies, peers prepare one Replication message per frequency (unless there are no keys belonging to the frequency) and apply the replication algorithm independently to each. Given that the number of frequencies is expected to be small in comparison to the number of keys, we expect the multiplication of Bloom filters per peer to present an acceptable overhead to the replication costs.

3.3.3 Random Walk Forwarding

FASE uses random walks for both the replication and retrieval of items. When implementing a random walk algorithm, there are multiple implementation choices,

which may differently impact the performance of the algorithm.

There are at least two possibilities to prevent random walks from visiting more than once the same node. One possibility is to store on the nodes the unique IDs of the messages that have visited him recently. When a message is received for the second time by a node, it is discarded. This approach may significantly impact the performance of algorithms relying on a small number of parallel random walks since there is the possibility that all the walks are discarded before finding some item. Note that the walk need not to be discarded, but in that case, there is a chance that the walk enters a cycle, thus wasting resources. The alternative is to include the list of traversed nodes in the message. Although it may lead to significantly large messages, this approach permits to nodes to avoid next hops that have been previously visited so that the probability of the message being dropped is reduced. We opted for the later for the replication process. REPLICATION messages are generated only once per frequency and per node and their length is bounded by the number of pointers to insert, so we expect the overhead to be acceptable. Furthermore, it ensures that replication messages are correctly forwarded until the replication is terminated (note that it is a single message that should not be terminated before storing all the pointers).

QUERY messages are more frequent than REPLICATION messages and can use multiple random walks. Storing every ID of the visited nodes in all the random walks could add a non-negligible cost to the transmission of queries. In this case we opted for a hybrid approach. Nodes store the IDs of the queries that visit them. We assume that, for nodes on wired networks, the storage cost of a circular queue with capacity for a large set of IDs is acceptable. When a node, n, receives a previously seen ID, it sends the query back to the last hop, n'. If this happens, n' forwards the query to any neighbor except n. This method alone could increase the search latency, for instance, if two nodes shared many neighbors and both were previously visited. In this case, the query could waste a few hops being forwarded from one node to the other. For this reason, we store a small set S of node IDs in the random walk message; its length should be set to a value that results in no more than a small increase of the QUERY message size. S contains the IDs of the more recently visited nodes. Random walks are not forwarded to any node belonging to S.

When using multiple random walks it is important to define a criteria that

permits to stop all the random walks after the first successful hit. In FASE implementation we used "adaptive termination" (Lv *et al.*, 2002) which dictates that when an item is found the query source is contacted and the walker is terminated. Walkers periodically establish a link from the hosting node to the source of the query to confirm that no reply was found and terminate otherwise.

3.4 Discussion

In this section FASE properties are enumerated, followed by a discussion of the optimal settings and assumptions covering two algorithm parameters: the number of pointers and the number of frequencies. The restrictions of the algorithm are discussed.

3.4.1 Properties

FASE was designed with fairness in mind, considering that all peers desire to have the least possible load imposed to them. An overlay where no participant has a role of higher importance is more able to cope with the peer transiency and possibility of failure faced by overlay networks in the Internet. The algorithm was designed to be scalable, in comparison to resource-intensive search algorithms for flat unstructured overlays such as the original Gnutella. Additionally, FASE should be efficient, that is, achieve a search latency comparable to other algorithms for unstructured overlays, while exhibiting a moderate resource consumption. Concerning these properties, from FASE architecture we can draw the following observations.

Fairness FASE enforces fairness by dividing the search space in equal parts and by storing pointers across all participants with a uniform distribution. Therefore, each participant performs a similar role on the overlay.

Scalability By using a fixed number of random walks, FASE can be more scalable than a flooding algorithm, particularly in the case of items that can only be found at more than a few hops away from the query source. The partition of the search space reduces the number of nodes that have to be searched on each query, equally contributing to reduce traffic on the overlay.

Efficiency The efficiency depends on the number of random walks used and the pointer replication. Higher replication rates will consume more resources at the replication phase and memory at the participants, but reduce the number of nodes that have to be searched to find an item. The same rationale applies to the number of random walks: more will result in a faster search but consume more resources (which is mitigated by the adaptive termination of the walks).

3.4.2 Number of Frequencies

The purpose of the replication algorithm storing one pointer per path, and of the search algorithm following paths in the key frequency, is that a pointer should be useful to several nodes (i.e. all the nodes close to the path where the pointer is) and not a single node. The rationale for this is that replicating pointers has a cost, thus pointers should be limited and their location optimized.

Given a number of frequencies $|\Phi|$ and the degree D of a constant degree overlay, the $\frac{D}{|\Phi|}$ ratio will have influence on the path length as it conditions the average number of frequencies reachable by each node. Since each hop traversed by a random walk on a node not belonging to the target frequency does not contribute to the success of the search or storage operation, each node should be aware of at least one node in each frequency so that it is able to route any random walk to a suitable location. Additionally, it is possible that some values of $\frac{D}{|\Phi|}$ exhibit a better use of pointers as, if nodes do not have any neighbor in the same frequency, the limited number of pointers would be stored with a skewed distribution (e.g. a pointer being stored every other node until the replica counter reaches 0). Since query messages follow a path expecting to find a pointer, if there are too many paths without pointers the search algorithm performance will be less than optimal.

Considering a $\frac{D}{|\Phi|}$ ratio that allows the formation of paths containing a high percentage of the nodes of the same frequency, FASE performance would suffer because large paths would have to be traversed to find or store a pointer. The pointer distribution would be affected as a replication message would traverse a possibly large number of nodes while storing only one pointer.

The evaluation of FASE will consider several $\frac{D}{|\Phi|}$ ratios and study its effect on the cost of replication and search.

3.4.3 Locality Sensitive Hashing

The hashing of keys, represented by strings, into a frequency space restricts the type of search in FASE to exact match queries. For instance, strings *kreutzer* and *krautzer* would probably yield a different frequency after being hashed by a function such as MD5 or SHA-1. Therefore a misspelled search, *krautzer*, for some item identified by the keyword *kreutzer* would fail unless a query visited a node with the matching item (recall that node's item indexes are always examined, regardless the query or node frequency, and that if the query is in the form of a string it can be matched to the corresponding item).

Although orthogonal to the dissertation, one possible solution to this problem is to use locality sensitive hashing (Indyk & Motwani, 1998) where, given a similarity function, similar objects are mapped to the same bucket; when a bucket has more than one element only one, chosen arbitrarily, is retained. Therefore, for some similarity metric, similar strings could be mapped to the same frequency.

3.5 Summary

In this chapter we described FASE, a search algorithm for unstructured overlays that combines a search space division method, a replication algorithm that disseminates pointers to items and a biased search algorithm based on random walks which leverages on the replication and search space division methods. As a high rate of churn characterizes P2P networks, we introduced a lightweight monitoring algorithm to maintain the ratio of pointers and thus improve the performance of FASE when subjected to churn. A discussion of FASE concluded this chapter.

Chapter 4

Evaluation

This chapter presents an evaluation of FASE and of the distributed monitoring algorithm (FASE+). The performance of FASE is compared with the Scalable Query Routing (SQR) algorithm. We start by describing the evaluation methodology, the churn model and the experimental parameters and then present and discuss the simulation results.

4.1 Methodology

We evaluated FASE through extensive simulations. FASE is first evaluated using a stable network, that is, a network were the nodes remain from the beginning to the end of the simulation. Afterward FASE and FASE+ (that is, FASE coupled with the distributed monitoring algorithm) are evaluated in a network subject to churn. Simulation results were obtained using PeerSim (Márk Jelasity, -) event driven simulation engine. The simulations use a 10,000 node network. For each test case we used 10 different topologies with constant degree, generated by HyParView (Leitão *et al.*, 2007), running multiple simulations. Plots present the aggregation of the multiple simulations and the standard deviation.

Each simulation creates 1000 unique data items, from a pool of more than 40000 items, each advertised by a distinct node selected at random. In a number of running P2P networks, items often have a non-uniform distribution that depends on their popularity. We opted for a uniform distribution model where each item is hosted by a single node, thus representing the least popular items of the networks. The rationale for this model is that the search for rare items is the worst case scenario and that an algorithm that shows good performance in the search for rare

items will show good performance in the search for more popular items, as popular items will be even more replicated. Furthermore, the search for rare items does not impose a higher maintenance overhead in the algorithms evaluated here.

For every algorithm evaluated, queries are performed for items that are present in the network, selected uniformly at random. In the case of a stable network, simulations have a deployment phase, where items are replicated, followed by a querying phase, where each node issues a query. In the case of a churning network, the querying phase is split in two: one *warm-up phase*, where items entering the overlay are replicated and nodes join and depart the network but no queries are performed, and a *search phase*. In the first half of the search phase, nodes arriving the network (and that have a session time greater than 30 seconds) perform a query for existing items; on the second half new items cease to be announced, to measure the effect of churn over old content.

The simulations are used to evaluate two metrics:

Hop Count the number of hops until an item is found.

Hop Limit the highest hop count for a given success rate, e.g. the hop limit for 90% is the highest hop count observed in the 90% most successful queries.

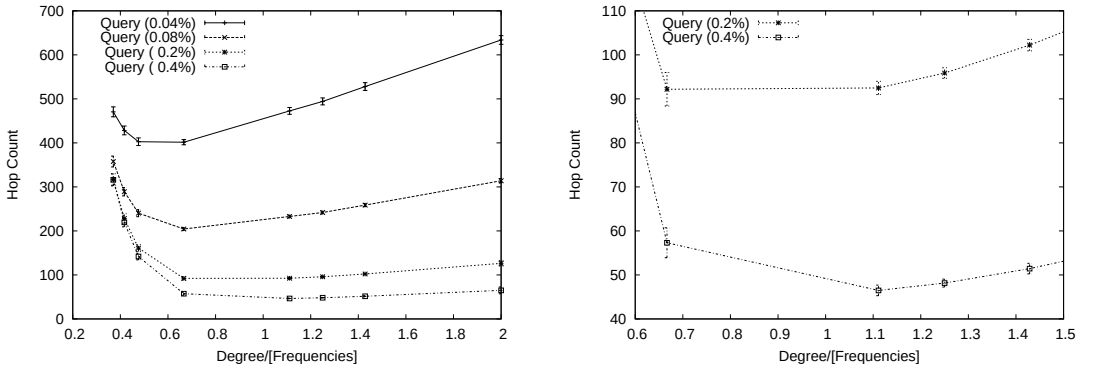
If multiple random walks are used, we observe the *message count*, that is, the number of messages effectively used by a query. Additionally, we observe the *failure ratio*, that is, the ratio of failed queries for items existing in the network. A query is considered failed when its random walks exceed their TTL without finding an item or when they cannot be forwarded; the TTL was set to the same size of the network.

4.2 Experimental Parameters

We observed FASE behavior for both low and high degree topologies, using different replication rates and varying the number of parallel random walks. The topology with $degree = 10$ models a network of participants with limited resources. With $degree = 30$ we have a topology which imposes a higher load on each user, due to the higher number of connections, to achieve a better hop count, as we will show in next section. We experimented with different number of frequencies until an optimal value was found. The "adaptive termination" (Lv *et al.*, 2002)

Table 4.1: Simulation parameters used by FASE.

Degree	10, 30
Number of frequencies	5, 7, 8, 9, 15, 21, 24, 27
Pointer Replication (#pointers)	0.04% (4), 0.08% (8), 0.2% (20), 0.4% (40)
Random Walks	1, 2, 4



(a) $\frac{D}{|\Phi|}$ and average hop count.

(b) $\frac{D}{|\Phi|}$ and average hop count for the highest pointer replication values.

Figure 4.1: Effect of the number of frequencies in the average hop count.

parameter of random walks (see Sec. 3.3.3) was configured so that each random walk checks for termination every four hops. Table 4.1 summarizes the parameters used.

4.2.1 Number of Frequencies and Degree

In Sec. 3.4.2 we referred that there should be an optimal value of $\frac{D}{|\Phi|}$, the ratio of the degree to the number of frequencies, that maximizes FASE performance while achieving an acceptable replication cost. In this section, we investigate how this ratio impacts performance.

We experimented several values of $|\Phi|$ over networks with $degree = 10$. Queries were performed using one random walk. Figure 4.1(a) depicts the average hop count in function of $\frac{D}{|\Phi|}$ for various replication rates and Fig. 4.1(b) depicts the same data for the replication rates that achieve the lower hop counts.

With the exception of the 0.04% replication factor, the higher values for the average hop count occur when $\frac{D}{|\Phi|}$ is lower than 0.5. The hop count increases for all the replication factors when the number of frequencies is much higher than the

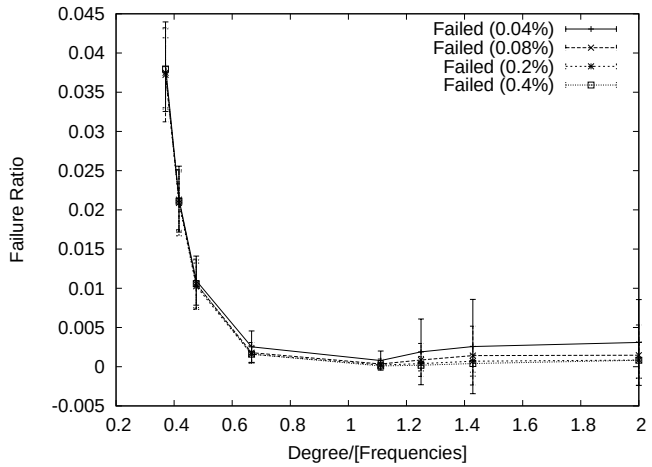


Figure 4.2: Failure ratio.

degree, in this case, when $|\Phi| \geq 21$. As each node is only aware of other nodes with, at most, 10 different frequencies and FASE performance depends on finding nodes with the same frequency of a query at a close distance, the hop count increases. If the pointers have a small replication factor (0.04%) the lowest hop count occurs when the ratio is between 0.7 and 0.5. In this case, the search algorithm benefits from a higher number of frequencies, as increasing $|\Phi|$ reduces the search space (which should be $\frac{|N|}{|\Phi|}$). However, the impact of higher replication rates in performance is greater than the increase of frequencies and, for the replication rates that achieve lower hop counts, from the tested values, the best results are achieved when $\frac{D}{|\Phi|} = 1.1$. The lowest number of failures occur for the same ratio, as it is shown in Fig. 4.2.

Figure 4.3 depicts the replication costs in function of $\frac{D}{|\Phi|}$ for various replication rates. The figure shows that for $\frac{D}{|\Phi|} = 1.1$, the hop count is 45% lower than in the worst case. As expected, the number of hops spent in replicating pointers grows as the probability of having multiple neighbors of the same frequency increases. As the neighbors of the same frequency increase, the probability of having a long path with a high percentage of the nodes assigned to some frequencies increases. Therefore, replication messages are required to traverse more hops before storing each replica. A replication message would traverse such a long path while storing only one pointer. While 1.1 is not the optimal $\frac{D}{|\Phi|}$ value for replication, for a process that will happen at most once per node and per frequency, it presents a

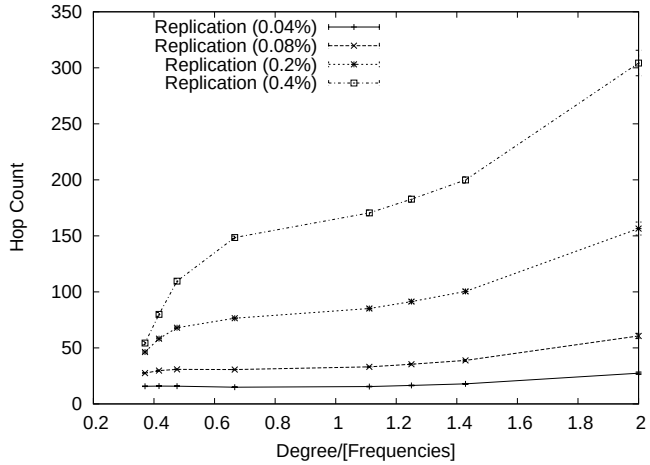


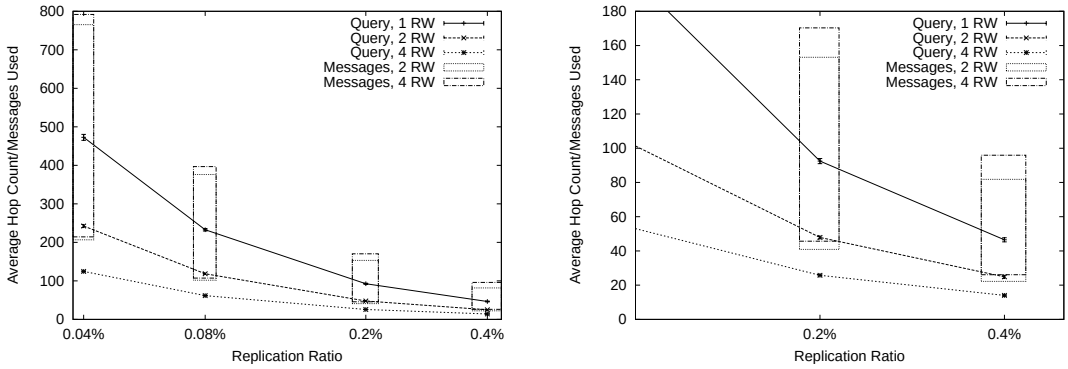
Figure 4.3: Effect of the number of frequencies in the replication costs.

good trade-off between the replication cost and query hop count. This is justified by observing Figs. 4.3 and 4.1, where it can be seen that, for replication rates that allow the query hop count to reach an acceptable number of hops, a ratio of 1.1 has the lowest hop count, and a replication cost approximately 50% below the ratio that consumes more replication messages. Additionally, a value of 1.1 has the lowest number of failed queries (Fig. 4.2).

4.3 Stable Network Evaluation

In this section we observe the behavior of FASE in a stable network. We present results for two degrees (10 and 30), using a $\frac{D}{|\Phi|}$ ratio of 1.1, that is, we used respectively 9 and 27 frequencies. Furthermore, we studied the hop count and number of messages used with multiple random walks. Unless otherwise noted, the data refers to the 10th degree topologies.

Figure 4.4(a) depicts the average hop count and the number of messages used (that is, the sum of the hops traversed by the multiple walkers) for the various pointer replication values and Fig. 4.4(b) zooms in on the same data for the 0.2% and 0.4% replication rates. The figure proves the advantage of using multiple random walks. When 4 random walks are used the hop count is 70% and 72% lower (respectively, with 0.4% and 0.2% pointer replication). However, due to the increased probability of finding an item and the adaptive termination, the total



(a) Average hop count and number of messages used.

(b) Average hop count and number of messages used for the highest pointer replication values.

Figure 4.4: Average hop count and number of messages used with multiple random walks (RW). Logscale on the x axis.

number of messages used only increases by 32% and 16%, when compared to the single random walk version. The standard deviation in the number of messages is high because the sum of the query hop count of the worst parallel random walks impose a heavy bias on the sample. This can be observed in Fig. 4.5, where the hop limit for 90% of the queries is presented.

Since the number of frequencies is greater with a higher degree, the number of nodes and keys per frequency is reduced. Therefore, with equal replication factors (that is, with a equal number of pointers), the use of a topology with a higher degree reduces the query hop count and the number of messages used, as is shown in Figs. 4.6(a) and 4.6(b). This presents an interesting trade-off: when deploying FASE, or a similar search algorithm, one has to weigh the cost of establishing and maintaining a high number of connections against the cost of search and of replication. In FASE, a topology with $degree = 30$, with a replication ratio of 0.04%, achieves lower query hop counts than with a replication ratio of 0.08% in a topology with $degree = 10$, since the lower degree topology has more pointers but also more nodes per frequency. Unless the cost of maintaining more connections during the lifetime of the node is higher than the cost of search, having a high degree is preferable.

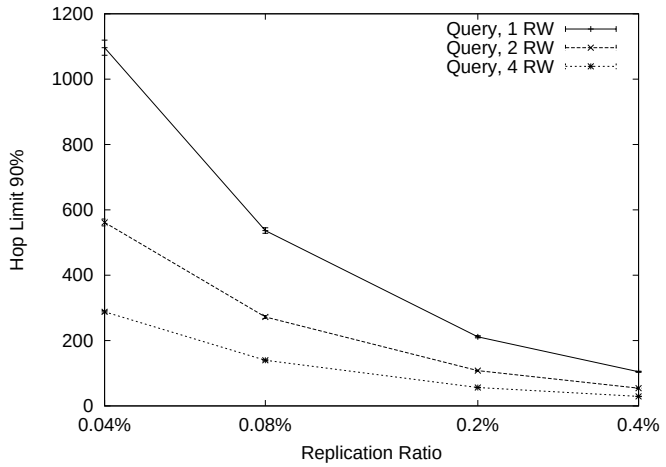


Figure 4.5: Hop limit for 90% of the queries. Logscale on the x axis.

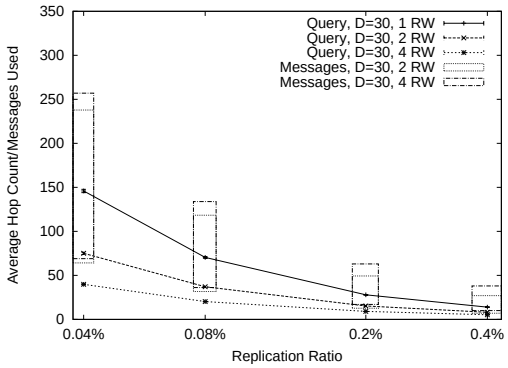
4.4 Churning Network Evaluation

The goal of this evaluation is twofold. First, to find if FASE is capable of sustaining the pointer loss and hop count degradation observable in FASE in the presence of churn. Second, to verify if the adopted churn mitigation strategy in FASE+ is adequate, particularly if it can use spaced enough updates to maintain a low overhead.

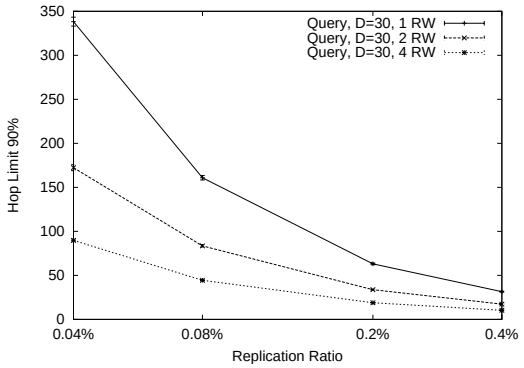
In the evaluation of an overlay subject to churn, FASE and FASE+ simulations use a $degree = 10$ network, four random walks and replicate pointers by 0.2% and 0.4%. This choice of parameters is justified by FASE results in the stable network evaluation, as it performed better using these values. It was shown in the previous section that using multiple random walks decreases the hop count with a moderate cost and that replication factors lesser than 0.2% result in high hop counts. The length of the simulation warm-up and search phases was set to 10800s. In the case of FASE+, the Δ parameter was set to 900s and 1200s.

We examined the average query hop count, the hop limit for 90% success and the query hop count over time. Additionally, in the case of FASE+, we observed the changes in number of pointers and backups during the simulations as well as the total number of items losing all their pointers during the simulation.

FASE does not clean up old replicas, however, due to memory constraints, a cleanup algorithm similar to FASE+ was executed during the simulations. Note that this not affect the evaluation of FASE, since all queries are for items that are



(a) Average hop count and number of messages used.



(b) Hop limit for 90% of the queries.

Figure 4.6: Average hop count, number of messages used and hop limit, using multiple random walks (RW) in a topology with $degree = 30$. Logscale on the x axis.

present in the network and, if a query finds a key but the item is no longer present, the result is discarded but not counted as failed.

4.4.1 Churn Model

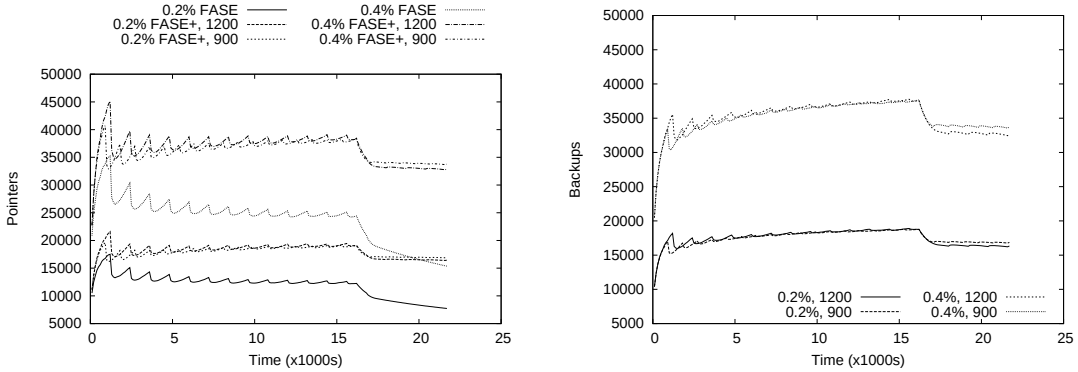
The constant arrival and departure of nodes will affect the evaluated algorithms in different ways. In the case of FASE, it will cause the loss of the replicated pointers and, in the case of SQR, it will out-date the routing tables of the nodes.

To simulate churn we opted to distribute the session length (the time between the arrival and departure of a node, see Sec. 2.2.1) of the nodes in the overlay according to the Weibull distribution using shape (k) and scale (λ) parameters for the three BitTorrent data sets (designated as "Red Hat", "Debian" and "FlatOut") studied by (Stutzbach & Rejaie, 2006). The values of k and λ are shown in Table 4.2. The "RedHat" and "Debian" sets have a similar session length distribution, although "Debian" has a slightly higher percentage of long lived peers. The "FlatOut" set has less extremely short and extremely long lived peers than the previous sets.

We simplify our model by maintaining the network size constant. When a node departs from the overlay another node joins, not necessarily in the same position, but establishing connections to nodes which are accepting new neighbors, thus contributing to maintain node degree. Nodes that depart from the overlay remain

Table 4.2: Session length distribution parameters.

Red Hat	$k = 0.34$, $\lambda = 21.3$
Debian	$k = 0.38$, $\lambda = 42.4$
FlatOut	$k = 0.59$, $\lambda = 41.9$



(a) Evolution of the number of pointers.

(b) Evolution of the number of backups.

Figure 4.7: Evolution of the number of pointers and backups in the RedHat data set.

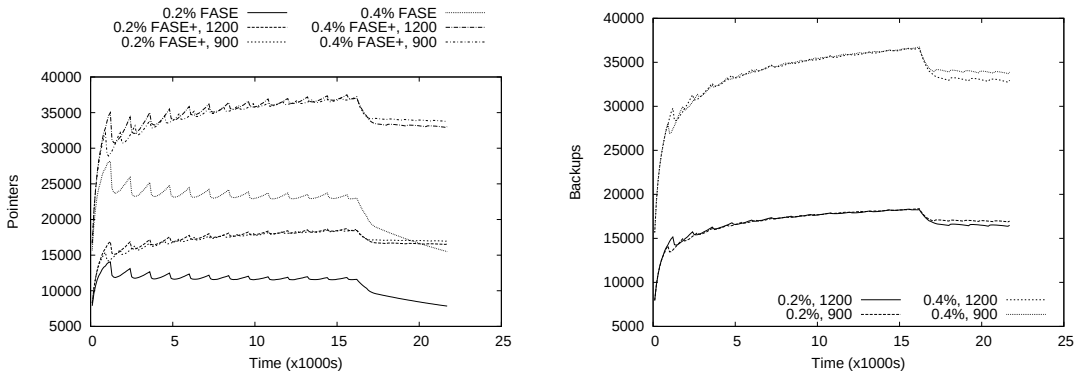
offline during the remainder of the simulation.

4.4.2 Pointer and Backups Evolution

In this section we evaluate the ability of FASE+ to maintain pointers under churn. The evolution of the number of pointers and backups was recorded with intervals of 100 seconds. A stale pointer cleanup algorithm is executed periodically to reduce the storage overhead during the simulation. As we only query for live items, this does not affect the results.

Figures 4.7, 4.8 and 4.9 show the evolution of the total number of pointers (for all items) over time, in FASE and FASE+, for the "RedHat", "Debian" and "FlatOut" sets, with update intervals of 900s and 1200s.

The peaks observed in the figures can be explained by the stale pointer cleanup algorithm. As they occur at times which match the Δ intervals counting from the simulation start. But for such peaks to happen, a high percentage of nodes should be timing out at the same time. There are two explanations for this. First, as Stutzbach et al. found in (Stutzbach & Rejaie, 2006), a large portion of peers are stable while short-lived peers constitute most sessions (as they have a high



(a) Evolution of the number of pointers.

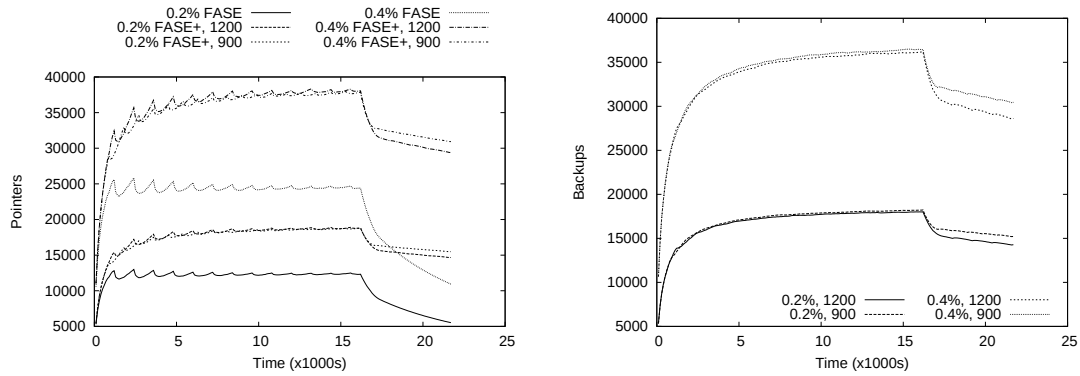
(b) Evolution of the number of backups.

Figure 4.8: Evolution of the number of pointers and backups in the Debian data set.

turnover rate). Therefore, the probability of a node at the start of the simulation to have a long uptime (relative to the length of the simulation) is not low. This is confirmed by the session time distribution of the peers. Second, the granularity of the snapshots (100s) which implies that every node that performs cleanup within that interval contribute to the peaks.

During the first half of the querying phase the patterns are similar although FASE+ is able to maintain a significantly higher number of pointers. Observe that the replica growth is due to nodes with new items entering the overlay and executing their initial replication algorithm. In the second half, after the announce of new items is stopped, both algorithms lose pointers, albeit FASE loses them at a higher rate whereas FASE+ is capable of maintaining the number of pointers after an initial decrease. The behavior of FASE+ in the second half of the simulation shows that the algorithm is capable of maintaining the availability of older items. Figures 4.7(b), 4.8(b) and 4.9(b) proves that the number of backups is kept on par with the number of pointers. Another interesting result is that the algorithm maintains the number of backups and replicas even when each update is spaced by 20 minutes.

FASE+ also reduces the number of live objects losing all their pointers. Figure 4.10 presents the cumulative count of objects that lost all their pointers present in the overlay. In the figure, the lines that present the results for FASE+ with various parameters overlap each other. As expected FASE with a replication factor of 0.2% loses more items than with a replication factor of 0.4%, as with more



(a) Evolution of the number of pointers.

(b) Evolution of the number of backups.

Figure 4.9: Evolution of the number of pointers and backups in the FlatOut data set.

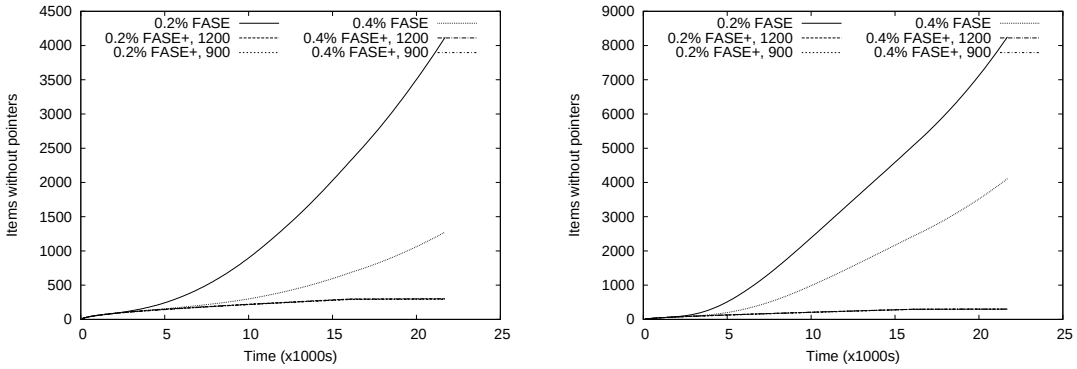
pointers there is a greater probability that some survive until the end of the simulation. FASE+ is able to substantially reduce the number of items that lose all their pointers.

4.4.3 Query Performance

The average hop count and the hop limit for 90% of the queries are presented in Fig. 4.11. Additionally, we observe in Fig. 4.12 and Fig. 4.13 the query hop count evolution in 10 minutes time slots. This allows us to observe the relation between the algorithm churn mitigation strategy and the hop count, both with unbiased queries, in the first half, and with queries biased to old items, in the second half.

Figure 4.11 presents the average hop count and the hop limit for 90% success for the RedHat and FlatOut sets, with an update interval of 1200s. As the RedHat and Debian sets present similar results, we opted to present only the RedHat results. In comparison with FASE, in the RedHat set FASE+ shows a lower average hop count, less 84% and 78% hops for, respectively, 0.2% and 0.4% replication rates. FASE+ shows a significant improvement in the hop limit, about 72% less for 0.2% replication and 70% for 0.4% replication. The percentages in the FlatOut set are similar. As FASE+ is able to maintain the number of pointers defined by the replication ratio, the queries take less time to find a key than in FASE, that loses pointers as an item age increases.

In the second half of the querying phase, FASE starts losing pointers of the



(a) Lost pointers in the RedHat set.

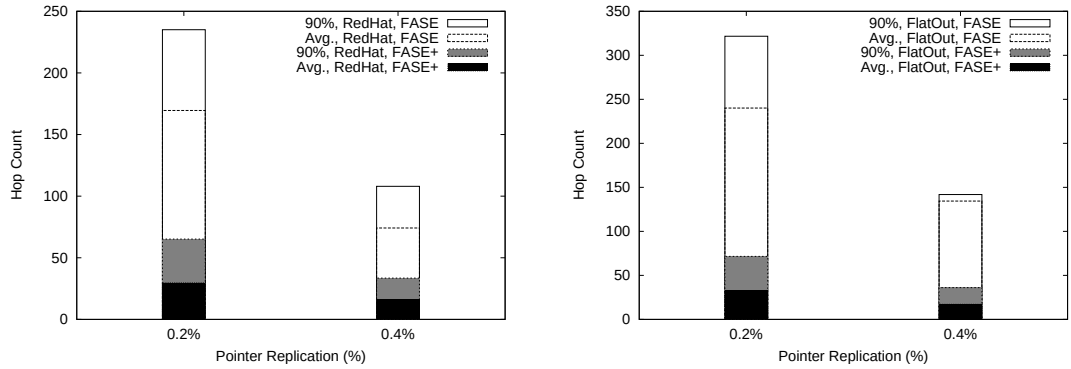
(b) Lost pointers in the FlatOut set.

Figure 4.10: Live items which lost all their pointers in the RedHat and FlatOut sets. The Debian set is similar to RedHat.

old items while FASE+ mitigates that loss. This is confirmed by the following figures, which plot the average hop count with the update interval set to 1200s. In Figure 4.12 we can see the effect of the loss of pointers in the query resolution. We omit the Debian set, due to the similarities with the RedHat set. While new items are being announced, FASE+ maintains the hop count values even when it increases in FASE; in the second half, where no new items are advertised and therefore, queries are only addressed to old items, FASE shows an increase while FASE+ is able to maintain the hop count. Figure 4.13 show the results of queries over time for the FlatOut set. The FlatOut distribution has less nodes with very short session times and less nodes with very long session times, and presents higher hop counts when compared with the other sets. However, FASE+ is able to maintain the hop count during both halves of the querying phase.

4.4.4 SQR

We implemented the Scalable Query Routing (SQR) algorithm (see Sec. 2.5.6), to be used as a comparison to FASE. SQR was chosen because it is an algorithm which aims at efficiency and scalability in unstructured peer-to-peer networks using a distinct approach from FASE. While FASE approach consists in dividing the search space and increasing the probability of an item to be found in a specific set of nodes, the SQR algorithm uses routing tables (which have to be maintained) at each node to guide query forwarding.



(a) RedHat (b) FlatOut

Figure 4.11: Average hop count and hop limit for 90% in the RedHat and Debian sets.

Table 4.3: Simulation parameters used by SQR.

Degree	10
Replication (#items)	0.01% (1), 0.04% (4), 0.08% (8), 0.2% (20), 0.4% (40)
d	8
m	40960
k	32

One could argue that, while FASE does not have the advantage of SQR, since SQRs probabilistic routing tables provide hints to where the queries should be directed, the pointer replication mechanism gives a better advantage to FASE in finding rare items. For that reason, test cases where SQRs contents are replicated will be presented.

Due to memory constraints imposed by its routing tables, SQR was evaluated only up to $degree = 10$ topologies. SQR's parameters d (the filter decay rate) and k (the number of hash functions) were selected based on the values used in (Kumar *et al.*, 2005) to achieve a low cost for the maintenance of the routing tables; the m parameter (the EDBF length) is lower than the value used in (Kumar *et al.*, 2005) as we used substantially less objects. Table 4.3 summarizes the parameters used in SQR simulations.

Figure 4.14 compares FASE and SQR average hop count. SQR is not suited to search for rare items: the average of the hop count for finding a unique item, the value for 0.01% replication ratio, is of 1360. As there is enough information to

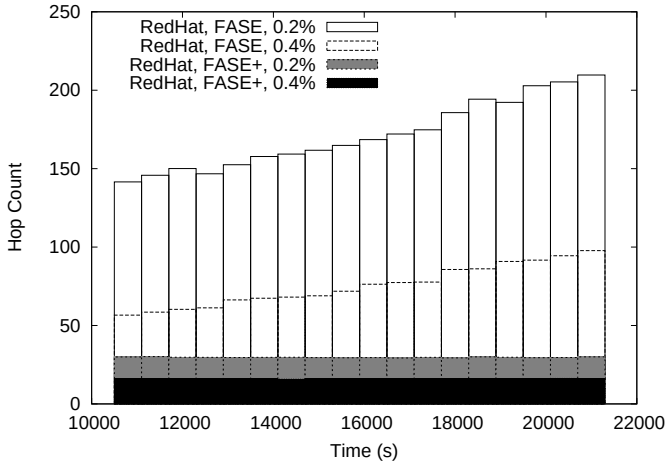


Figure 4.12: Query hop count split in 10 minutes time slots for the RedHat set.

route the query only at a close distance from the item, depending from the decaying factor, the queries are forwarded in a random fashion most of the time. This results seems to point that replication may be more adequate than building routing tables as a strategy to increase the performance of unstructured overlays. We tested SQR with its contents replicated with the same ratios observed in FASE pointer replication. With one random walk and without the benefit of routing information, FASE results are close to SQRs. This proves the benefit of the division of the search space in multiple frequencies. Since the replication is equal, this division is the only leverage of FASE in this case. The hop limit for 90%, presented in Fig. 4.15, follows the same pattern of Fig. 4.14.

SQR does not perform well in overlays subject to churn. Like in the stable network scenario, we first tested SQR with a unique item and set the interval of filter dissemination to 60s. Then we compared SQR to FASE+ with a replication ratio of 0.2%, using a single random walk and an update interval of 1200s in FASE+. Table 4.4 presents the results for the RedHat set. Without replication the failure ratio, that is, the number of queries that exceed their TTL (set to the size of the network) is above 50%. Churn emphasizes the random behavior seen in the stable simulations, leading to query hop counts that exceed the TTL. With 0.2% replication SQR performance increases, but it is significantly lower than in stable conditions (from, approximately, 120 to 1580 hops); FASE+ shows a performance under churn as good as SQR in a stable network, in the same conditions.

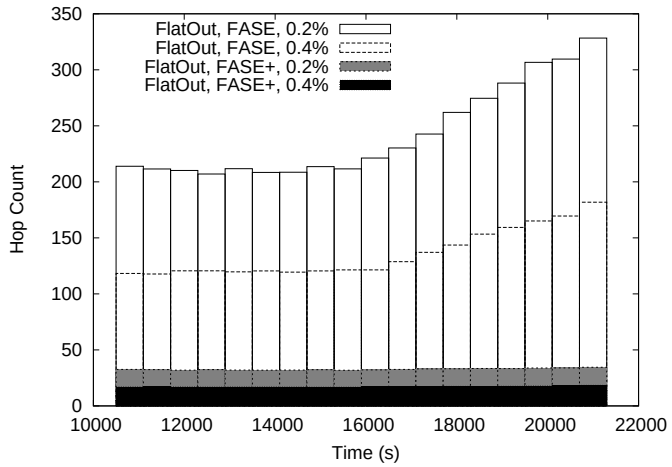


Figure 4.13: Query hop count split in 10 minutes time slots for the FlatOut set.

Table 4.4: FASE+ and SQR subject to churn.

	Hop Count	Failure Ratio
FASE+ (0.2%, 1 random walk)	108	0
SQR (no replication)	2227	0.0560
SQR (0.2%)	1584	0.001

4.5 Summary

This chapter presented the evaluation of FASE and of the distributed monitoring algorithm. In the first part of the evaluation, the test methodology was defined and the parameters that achieved an adequate trade-off between the replication cost and search efficiency were discussed. During the remainder of the chapter, the algorithms were tested and compared to the SQR algorithm in stable and churning networks.

Evaluation showed that FASE presents a valid search strategy. Additionally, they suggest that increasing the availability of items through replication can be a better approach for finding rare items when compared to the use of routing information. It was proved that FASE+ successfully mitigates the effect of churn in FASE and that it is able to cope with varying levels of churn.

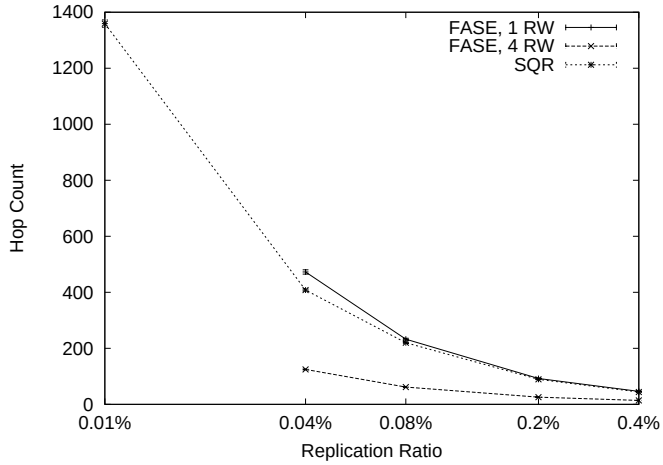


Figure 4.14: Average hop count for FASE and SQR. Logscale on the x axis.

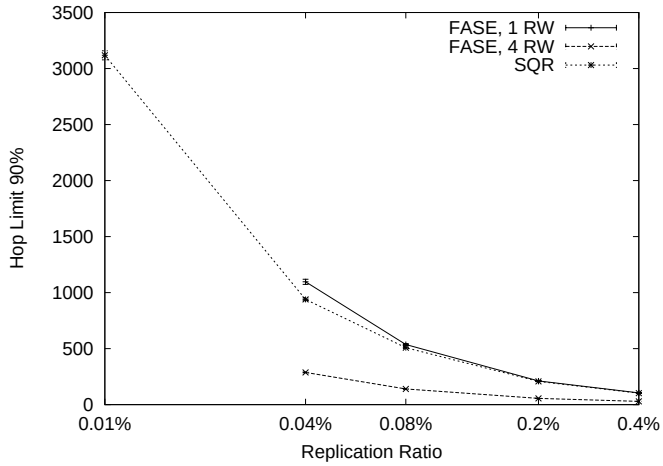


Figure 4.15: Hop limit for 90% success for FASE and SQR. Logscale on the x axis.

Chapter 5

Conclusion and Future Work

Attaining scalability, low resource consumption and low query latencies in P2P systems based on unstructured overlays is challenging. Search protocols such as the original Gnutella are able to locate content within a low number of hops, but use resource intensive, unscalable, algorithms. In order to scale, some protocols build a topology organized in two tiers, with regular and supernodes. The supernode function can be described as a "hub", connecting regular nodes, indexing the regular nodes contents and forwarding queries to other supernodes. In this approach, the supernodes have to offer much more resources than their regular counterparts, a feature that may not be desired by some users.

Search algorithms that have a moderate resource consumption, such as random walks, may require a large number of hops to locate a resource, unless hints for the best location are provided. However, hints may rapidly become out of date due to the constant arrival and departure of nodes that characterizes P2P overlay networks. There is no perfect solution that combines the lowest resource consumption and query latency and the highest scalability, as structured overlays are scalable but costly to maintain and have a unpredictable performance when subjected to churn, and unstructured overlays search methods that achieve better query performance are usually unscalable and resource intensive.

In this dissertation, we presented the Frequency-Aware SEarch (FASE) algorithm, a search protocol over unstructured, non-hierarchical overlays. FASE uniformly partitions the nodes and the keys used to locate items in a common frequency space. The keys are replicated in the nodes that share their frequency. Queries are mappable into the frequencies of the searched keys and directed to the

corresponding nodes. FASE does not impose membership constraints to the nodes in the overlay.

Evaluation showed that FASE can achieve low hop counts when locating resources in low degree networks where every participant contributes with similar resources. The efficiency of FASE depends on the balance of the cost of replication, which happens once in the lifetime of an item, and search, which can be repeated many times. If an item is to be searched for often, the replication costs will be progressively lower, as the existence of more pointers, which implies a higher replication cost, reduces the cost of all the queries.

FASE+, a distributed replica monitoring algorithm for FASE was implemented. It was proved that this algorithm sustains FASE performance for different rates of churn with a moderate additional cost.

FASE was compared with the Scalable Query Routing (SQR) algorithm. It was shown that, in a stable overlay, under similar conditions FASE performance is close to SQR. However, when the overlay is subjected to churn, FASE performs better than SQR. The results suggest that, to increase the availability of items (particularly, rare items), a replication policy is more adequate than the building of routing indexes.

5.1 Future Work

FASE increases the availability of items using a replication policy. It was shown that this policy is specially effective at improving the availability of rare items and more effective than using routing tables (as in the SQR algorithm). However, when searching for popular items, the validity of both approaches can be questioned as both imply some overhead and any random search would find the items with some good probability. SQR could avoid indexing popular items and its search could be improved using multiple random walks, but it would not be able to leverage on the search space partition as FASE does. A possible solution to estimate the popularity of items and, based on that estimate, to adapt the number of pointers disseminated by FASE, is to passively observe the query message traffic to find the more frequent search keys (as the popularity of the searches has a close relation to the availability of the items). This estimate does not need to be accurate but to fall within the same order of magnitude.

FASE+ performs well when using distanced monitoring intervals, but its resource usage could be improved. For instance, if a node holding a pointer and the node holding its backup happen to be neighbors, there is no need to wait for the update interval to pass (or to establish an additional connection to check the liveness of the pointer or backup); in this case nodes could detect failed pointers or backups at the same time they detect membership changes.

We presented an evaluation of FASE based on the message costs and assuming a constant degree network. The performance of the algorithm in different topologies as well as an evaluation of the bandwidth and the load at each pear, should receive further attention in the future.

Bibliography

BBC (2008). BBC iPlayer Help - What is peer-to-peer? Http://iplayerhelp.external.bbc.co.uk/help/download_programmes/peer2peer, viewed at 8 Jun 2008.

Bloom, B.H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, **13**, 422–426.

Broder, A. & Mitzenmacher, M. (2004). Network Aplications of Bloom Filters: A Survey. *Internet Mathematics Vol. I, No. 4: 485-509*.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A. & Gruber, R. (2006). Bigtable: A distributed storage system for structured data. *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*.

Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N. & Shenker, S. (2003). Making Gnutella-like P2P Systems Scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, 407–418, ACM Press, New York, NY, USA.

Chen, J., Ramaswamy, L. & Meka, A. (2007). Message Diffusion in Unstructured Overlay Networks. *Sixth IEEE International Symposium on Network Computing and Applications, 2007. NCA 2007.*, 126–133.

Cholvi, V., Felber, P. & Biersack, E. (2004). Efficient Search in Unstructured Peer-to-Peer Networks. In *SPAA '04: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 271–272, ACM, New York, NY, USA.

Clarke, I., Sandberg, O., Wiley, B. & Hong, T.W. (2001). Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, **2009**, 46–66.

Clip2 (2001). The Gnutella Protocol Specification v0.4. Http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

Fessant, F., Handurukande, S., Kermarrec, A.M. & Massoulié, L. (2004). Clustering in Peer-to-Peer File Sharing Workloads. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS), San Diego, USA. (2004)*.

Garbacki, P., Epema, D.H.J. & van Steen, M. (2007). Optimizing Peer Relationships in a Super-Peer Network. In *27th International Conference on Distributed Computing Systems (ICDCS 2007)*, Toronto, Canada.

Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M. & Zahorjan, J. (2003). Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. *SIGOPS Oper. Syst. Rev.*, **37**, 314–329.

Indyk, P. & Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 604–613, ACM, New York, NY, USA.

Kumar, A., Xu, J. & Zegura, E. (2005). Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks. *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, **2**, 1162–1173.

Leitão, J., Pereira, J. & Rodrigues, L. (2007). Hyparview: a membership protocol for reliable gossip-based broadcast. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 419–429, Edinburgh, UK.

Liang, J., Kumar, R. & Ross, K. (2005). The Kazaa Overlay: A Measurement Study. *Computer Networks (Special Issue on Overlays)*.

Lv, Q., Cao, P., Cohen, E., Li, K. & Shenker, S. (2002). Search and Replication in Unstructured Peer-to-Peer Networks. In *ICS '02: Proceedings of the 16th International Conference on Supercomputing*, 84–95, ACM, New York, NY, USA.

Márk Jelasity, G.P.J., Alberto Montresor (-). PeerSim: A Peer-to-Peer Simulator. Http://peersim.sourceforge.net.

Maymounkov, P. & Mazières, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 53–65, Springer-Verlag, London, UK.

Qiao, Y. & Bustamante, F.E. (2006). Structured and unstructured overlays under the microscope: a measurement-based view of two p2p systems that people use. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, 31–31, USENIX Association, Berkeley, CA, USA.

Rhea, S., Geels, D., Roscoe, T. & Kubiatowicz, J. (2004). Handling Churn in a DHT. In *Proceedings of USENIX'04 Annual Technical Conference*.

Saroiu, S., Gummadi, P. & Gribble, S. (2002). A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking (MMCN '02)*.

Sripanidkulchai, K., Maggs, B.M. & Zhang, H. (2003). Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *INFOCOM*.

Stutzbach, D. & Rejaie, R. (2006). Understanding Churn in Peer-to-Peer Networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 189–202, ACM, New York, NY, USA.

Stutzbach, D., Rejaie, R. & Sen, S. (2005). Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. In *IMC'05: Proceedings of the Internet Measurement Conference 2005 on Internet Measurement Conference*.

Tsoumakos, D. & Roussopoulos, N. (2003). Adaptive Probabilistic Search for Peer-to-Peer Networks. In *P2P '03: Proceedings of the 3rd International*

Conference on Peer-to-Peer Computing, IEEE Computer Society, Washington, DC, USA.

Yang, B. & Garcia-Molina, H. (2002). Improving Search in Peer-to-Peer Networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, 5–14.